A Service-Oriented ZigBee Gateway Based on the OSGi Framework

Stefano Chessa, Francesco Furfari, Michele Girolami, Stefano Lenzi, Francesco Potortì

***Abstract*—The ZigBee standard defines a service-oriented framework for the implementation of Wireless Sensor Networks (WSNs). In the recent years ZigBee has received the attraction of many research studies focused on the design and the implementation of network gateways able to access and to interact with ZigBee sensors from heterogeneous networks.**

To address the interoperable challenge, this paper presents ZB4OSGi, an OSGi-based ZigBee gateway able to export the ZigBee network services in the OSGi execution environment without requiring any prior knowledge about the ZigBee protocol. ZB4OSGi exploits a 3-layered architecture enabling the access, abstraction and integration of ZigBee devices to different channels like UPnP or SOAP-REST Web Services.

The paper faces with a concrete implementation of ZB4OSGi, by highlighting the features with respect to already existing ZigBee gateways.

***Index Terms*—ZigBee gateway, context-aware services, Middleware, Wireless Sensor Networks.**

I. Introduction

IN the recent past a consortium of major industries, interested in Wireless Sensor Networks (WSNs) [1], delivered a new industrial standard called ZigBee [2,3]. ZigBee is built on the IEEE 802.15.4 stack (which specifies physical and MAC layer of low-power WSN) and defines a service-oriented framework for the realization of WSN applications. ZigBee provides multi-hop communication mechanisms and basic strategies for the realization of service-oriented WSN applications. Its main application fields are home and factory automation, consumer electronic and healthcare.

Interacting with a ZigBee network requires prior knowledge about the ZigBee protocol, in particular the messages (frames) format, the interaction paradigm, and the ZigBee clusters and profiles. The possibility of accessing to the ZigBee network without such prior-knowledge and from heterogeneous networks, represents a challenging task.

Such kind of interaction requires to design and to implement ZigBee gateways able to ease the access to the ZigBee nodes and simultaneously able to export the ZigBee services to different target networks. More generally, the design of a ZigBee gateway gives rise to two main aspects that should be taken into account:

1) seamless integration: ZigBee nodes become accessible from outside, without any prior knowledge about the specific technology (message format, hardware features, interaction paradigm, network topology etc.).
2) Interoperability: services exposed by the ZigBee nodes cooperate by adopting a service-oriented model. The services can be integrated within existing architectures, drawing the so-called mash-up services [24].

Several ZigBee gateways have been already proposed, but some important limitations still are present: (i) most of the gateways only rely on specific ZigBee hardware, without providing any abstraction layer able to generalize from the ZigBee node, (ii) most of the gateways convert the ZigBee frames to only one specific target technology and only few of them aim at the generalization of the target network, and,. finally, (iii) not all the ZigBee gateways recognize ZigBee devices adhering to the ZigBee standard profiles or let custom ZigBee devices be accessed from the gateway itself.

This paper presents an OSGi-based ZigBee gateway (hereafter called ZB4OSGi), that takes into account the previously described aspects and limitations of the existing solutions. ZB4OSGi exports the ZigBee network services via different channels by exploiting a 3-layered architecture. In our approach, the gateway exports an abstract view of the ZigBee network, in which only the services provided by the ZigBee nodes are mapped into some OSGi services. In turn, exploiting the potentials of the OSGi execution environment, such services can be dynamically exported by means of different application-level technologies (for instance UPnP protocol, SOAP/REST-based services and others).

ZB4OSGi fully integrates ZigBee nodes adhering to the standard ZigBee Home Automation Profile, but further extensions to the set of profiles can be easily applied. Moreover ZB4OSGi implements a hardware abstraction layer that lets heterogeneous ZigBee hardware be used as network entry-point.

ZB4OSGi has been released under open source license in [22] as project of the AALOA open association [21] and a development team currently maintains the project with periodical updates, news and bug fixes.

The rest of the paper is organized as follows. Section II introduces the OSGi platform and the ZigBee stack as preliminary concepts; section III presents the state-of-the-art for the ZigBee gateways with a comparison table among the reviewed ZigBee gateways. Section IV introduces the ZB4OSGi solution; section V describes every layer of the ZB4OSGi architecture. Section VI provides a case study description, while section VII draws some conclusions.

Francesco Furfari, Michele Girolami, Stefano Lenzi, Francesco Potortì are with Istituto di Scienza e Tecnologie dell'Informazione – National Research Council, via Moruzzi 1, Pisa, Italy (e-mail: francesco.furfari@isti.cnr.it, michele.girolami@isti.cnr.it, stefano.lenzi@isti.cnr.it, francesco.potorti@isti.cnr.it).

Stefano Chessa is with Istituto di Scienza e Tecnologie dell'Informazione – National Research Council, via Moruzzi 1, Pisa, Italy and with Department of Computer Science, University of Pisa, Largo Pontecorvo 3, Pisa, Italy (e-mail: ste@di.unipi.it).

II. Preliminary Concepts

A. OSGi model

The Open Source Gateway initiative specification (OSGi) [9] defines a service oriented, component based platform for Java developers, and it offers a standardized way to manage the software life cycle. The OSGi implementations are containers running on top of a Java virtual machine, in which components can be installed, removed, started, and stopped at run time. An OSGi component (called bundle) is a JAR file that contains Java classes, resources and metadata describing the dependencies with other bundles. The main features that OSGi offers are:

1) a service model where every application component can be registered as service into a service registry.
2) An execution environment where multiple applications can run on the same virtual machine.
3) A set of API for the control of the bundles life cycle.
4) A secure environment where multiple applications can coexist without affecting each other.
5) A cooperative, distributed environment where bundles can discover each other independently of their hosting platform.

The bundles wishing to detect the presence of a particular service configure a service listener (with appropriate filters) and, as soon as the specified service becomes available, the OSGi framework notifies all the listeners with a service handler instance.

Although OSGi was initially thought as a platform supporting gateways, it became popular also in other fields. For example, OSGi also meets the requirements for pervasive spaces and smart environments as observed in [10].

B. ZigBee

The ZigBee specification defines the network and application layer of low-power wireless networks based on the IEEE 802.15.4 [2, 3] standard.

The network layer provides support to star, tree, and peer-to-peer multi-hop network topologies. At this layer, each node (or ZigBee node) is a physical component identified by a 16 bits network address. The network layer provides services for the initialization of the network, nodes addressing, multi-hop routing, packet forwarding and management of connections and disconnections of nodes. At the network layer the nodes can be either end-devices or routers. End-devices do not have any routing capabilities, rather, when they join the network they connect to a router and rely on it for all their communications. One routeracts as network coordinator and takes the address 0. Its role is to create the network and to define the address space. In a typical configuration, the network has a tree topology rooted in the coordinator. An overview of the ZigBee protocol stack is shown in Figure 1.

The application layer provides a framework to support, configure and manage distributed applications. The application layer comprises the Application Framework, the ZigBee Device Object (ZDO) and the Application Support Sublayer (APS), which offers functionalities of a transport layer. The Application Framework contains a number of Application Objects (APO), i.e. user defined application modules (also called application-level devices) that implement a ZigBee application (or at least sub-component of a distributed ZigBee application). The ZDO provides services that allow the APOs to organize themselves into a distributed application. The APS provides data and management services to the APOs and ZDO.

Each APO is associated to an EndPoint (EP) of the Application Framework, and it is univocally identified by the network address of the hosting network-level device and by the EP number (that ranges from 0 to 240, in particular EP 0 is reserved to the ZDO). An application framework can host up to 240 APOs.

To enable interoperability of nodes from different manufacturers, the ZigBee alliance defines the concepts of clusters and application profiles. A cluster is an application message containing one or more attributes. Clusters are defined in a separate specification of the ZigBee alliance, the ZigBee Cluster Library (ZCL). In general, an APO supports a collection of clusters. The application profile is a collection of device descriptions that form a cooperative application. For example the Heating Application Profile provides the description for the Thermostat device and the Furnace device. Hereafter we will refer to ZigBee nodes only as hardware devices, while we refer to ZigBee devices as hardware devices hosting the full ZigBee stack and ZigBee applications.

The ZDO provides to the APOs device and service discovery. Device discovery allows an APO to obtain the network address of other network nodes. The routers respond to the device discovery by returning their addresses and the addresses of all their associated end-devices. The service discovery exploits cluster descriptors and cluster identifiers to determine the capabilities offered by a given APO. An APO can inquiry about the capabilities provided by a specific APO or use a matching mechanism to detect the presence of APOs providing a given capability.

ZigBee also defines a binding mechanism among APOs. By means of the binding, two or more APOs can be connected with each-other. In this way whenever an APO

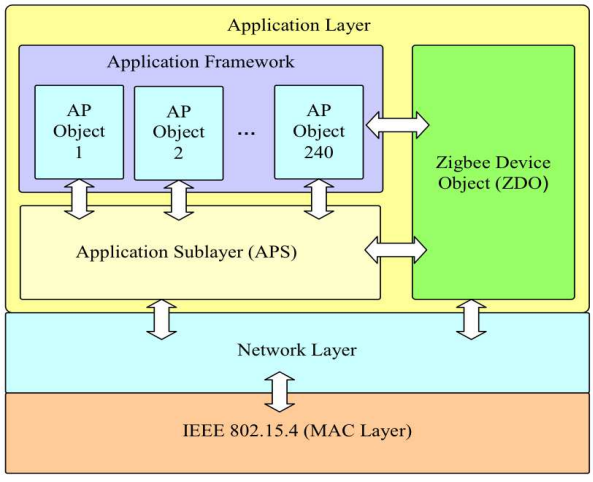

Fig.1. The ZigBee stack

emits a message with a specific cluster identifier, the message is automatically routed to a set of APOs according to the binding table.

The ZDO provides mechanisms for the management of the binding table; such mechanisms simplify the way in which the devices address themselves and it can be used to

TABLE I
COMPARISON AMONG THE ZIGBEE GATEWAYS

Criteria	[12]	[13]	[14]	[15]	[16]	[17]	[18]
Gateway architecture.	Component	Component	Component	Component	Component	Mixed	Service
Integration mechanisms	Knx-ZigBee integration	UPnP integration	UPnP integration	Protocol translation ZigBee - Ethernet	Web-server integration	Web-server, Mobile app Web-service integration	OSGi service integration
Hardware abstraction.	The gateway relies on RadioPulse ® stack. No hardware abstraction provided.	No specific hardware adopted. No hardware abstraction provided.	The gateway relies on Texas Instruments® Evaluation board. No hardware abstraction provided.	The gateway relies on custom hardware. No hardware abstraction provided.	The gateway relies on custom hardware. No hardware abstraction provided.	The solution relies on XBee® nodes. No hardware abstraction provided.	The solution relies on custom hardware. No hardware abstraction provided.
Device abstraction.	Support for Zigbee HA profile	Support for ZigBee HA profile	Support for ZigBee HA profile	No ZigBee profile supported	No ZigBee profile supported	No ZigBee profile supported	Support for ZigBee HA profile
Extension mechanisms	No support for custom cluster.	Support for custom cluster only mentioned as future work.	No support for custom cluster.	No support for custom cluster.	No support for custom cluster.	No support for custom cluster.	Support for device implementing custom cluster.

enable automatic eventing mechanisms. For example the ZCL exploits the binding to implement attribute reporting, the notification of alarms and more generally all notification capabilities.

III. RELATED WORKS

The initial ZigBee specification did not define any notion of gateway for the ZigBee networks, only in late 2010 the ZigBee alliance delivered the specification for the ZigBee gateway [8]. This specification defines a gateway for the interconnection of ZigBee and IP based networks. The approach used in this specification is to expose a web service (based on SOAP/REST or, alternatively, adopt the GRIP protocol), through which applications in the IP network can inject queries into the ZigBee network and receive responses. In addition, the gateway exposes functions to access all the layers and components of the Network layer and Application framework (including the ZDO and APS) of the ZigBee-side of the gateway. With this approach the queries to the ZigBee network are expressed in terms of XML schemas, and they are translated into the appropriate cluster by the ZigBee gateway. As a consequence, the applications wishing to interact with the ZigBee devices should be aware of the details of the ZigBee protocol (including address of the node, fields of the cluster etc.). This approach is rather different from the one we used to design ZB4OSGi, since in our proposal the gateway exposes refined services for each ZigBee device it discovers. Hence, the applications do not have to discover ZigBee nodes and consequently they do not need to know the low-level protocol details.

Gateways of ZigBee have also been subject of intensive, independent studies in the recent past [12-18]. Some of these works are mainly addressed to the design and implementation of gateways for protocol translation. Fits in this case the work presented in [12] where a gateway between Konnex (KNX) and ZigBee is presented. This work proposes a solution where the gateway translates KNX telegrams into ZigBee frames and vice-versa. The implementation relies on a multi-component gateway able to define such a mapping, but it does not provide any general integration solution for the node interaction or the hardware abstraction.

The solutions provided in [13, 14] are both focused on the implementation of UPnP and DLNA – ZigBee gateway. In [13] is proposed a solution aimed at the integration of ZigBee into DLNA networks by means of gateways. One of the components of the gateway is responsible for creating virtual UPnP devices as soon as it acquires relevant information on the ZigBee network. In a similar way, it creates virtual ZigBee application objects for every UPnP device found. This approach has the limitation that the gateway must be the ZigBee network coordinator. Furthermore, to our understanding, it does not addresses abstraction of ZigBee nodes. Similarly, in [14], the authors implement an internetworking gateway between UPnP and ZigBee focusing on the discovery mechanisms. The authors present a gateway based on two main components: UPnP - ZigBee gateway (UZG) and the ZigBee network topology manager. The UZG is made of the Application Object Manager, the ZigBee Device Manager (ZDM) and the Virtual UPnP Proxy Manager (VUPM). The ZDM controls the ZigBee nodes, it reflects all the relevant changes to the VUPM. The VUPM is responsible for creating or removing Virtual UPnP Proxy (VUP) as soon as some events occur on the ZigBee network. The authors provide a mapping between the meta-data of the ZigBee devices and the ones of VUP. With respect to [13], here it is discussed how the ZigBee devices are abstracted by the VUPs, however both of the solutions ([13, 14]) strictly rely on specific ZigBee hardware without introducing any hardware abstraction layer.

A different approach is adopted in [15], where a gateway for protocol translation is described. The author provides a mapping for all the 802.15.4/ZigBee protocol layers to the corresponding Ethernet ones. Differently than our approach, this work does not focus on ZigBee device abstraction or hardware abstraction mechanisms. In fact, this approach requires the users to implement their own software application in order to exploit the protocol translation and

interact with the ZigBee nodes. A web-oriented solution is described in [16]. The paper proposes a web-sensor gateway able to let the ZigBee network be accessible by means of a standard HTML browser. Similarly to [15], the gateway provides a protocol translation from ZigBee to Ethernet. The author describes also a web-server able to generate dynamic web pages reflecting the ZigBee network. This solution requires that the users own a topological perspective of the ZigBee network, addressing each individual nodes in the sensor network.

In [17] a vertical solution based on SmartBee is presented. The SmartBee specification lets the ZigBee nodes be accessible via multi-channel solutions, e.g. by means of regular web interface, mobile application or web service standard specification. The work presents some notable aspects, however the whole solution relies on the SmartBee specification build on top of the MAC layer of ZigBe, and being adherent to the SmarBee specification represents a non-negligible requirement for a general-purpose solution. Neither this work addresses on hardware abstraction.

The approach proposed in [18] caught our attention since it goes in the direction of representing ZigBee devices as OSGi services. In this work the gateway acts as the ZigBee coordinator, and it assumes that the nodes in the network periodically announce their presence by sending, to the coordinator, their ID and profile. As the ZigBee coordinator detects the presence of a new node in the network, it downloads a software service for that ZigBee device (wrapped within an OSGi bundle) and registers the service in OSGi. With respect to this work, our approach makes a further step of abstraction for the ZigBee devices. The ZB4OSGi gateway exposes the ZigBee devices on one or more access technologies available on the OSGi platform. Furthermore we remove the assumptions that the gateway acts also as the ZigBee coordinator and that the nodes have to periodically announce themselves (currently defined as optional feature). Finally, with our solution, we produce a different level of abstraction of the ZigBee devices. For example we can further refine the ZigBee devices with OSGi services that aggregate data produced from the ZigBee network, compose multiple ZigBee nodes as one single virtual node, hide the ZigBee network topology etc.

The result of the analysis of the state-of-the-art is summarized with Table I. The Table reports on the rows a list of important features that characterize the reviewed solutions. Currently we identify the following features:

1) Gateway architecture: service-oriented approach where the ZigBee devices are modeled in terms of exported services, multi-component approach where the ZigBee devices are accessed interacting with a specific component of the gateway, mixed solutions.
2) Integration mechanisms: how the ZigBee network can be accessed from outside network and which technologies can be integrated.
3) ZigBee hardware abstraction: how the gateway abstracts from specific ZigBee hardware.
4) ZigBee device abstraction: how the gateway manages ZigBee devices adhering to the standard ZigBee profiles.
5) Extension mechanisms: how the gateway manages the extendibility of the ZigBee clusters for custom behavior of the ZigBee Devices.

IV. ZB4OSGI GATEWAY

A. Architecture Guidelines

The main purpose of the ZB4OSGi gateway is to provide a simplified access to the functionalities offered by ZigBee devices. We aim at offering intuitive interfaces that link between those functionalities and external applications.

The ZB4OSGi gateway has been designed keeping in mind the following guidelines:

Dynamic discovery of ZigBee nodes: ZB4OSGi exploits the discovery mechanisms of ZigBee. As soon as new ZigBee nodes join the network, ZB4OSGi reacts to these events by registering new OSGi services reflecting the implemented APOs.

Abstraction of ZigBee devices: ZB4OSGi not only recognizes ZigBee devices adhering to the ZigBee profiles (e.g. On/Off Switch device, Remote Control device, Light Sensor device), but it also abstracts them. This allows the high-level applications to ignore the notion of cluster and message format (deeply connected with ZigBee terminology), and to focus only on data gathered from the nodes with the adoption of more intuitive APIs. This approach is rather different from the solution adopted by the ZigBee alliance (see section III), where the gateway specification offers an interface that enables to send clusters within the ZigBee network. Although such interfaces are exposed in terms of web services (SOAP or REST), the external applications should be aware of the internal protocols of ZigBee and use the clusters accordingly.

Extension mechanisms for ZigBee devices: the ZigBee Cluster Library specification defines an extended set of clusters to be used with the ZigBee devices. However, to meet additional requirements, it includes also a mechanism that enables the development of custom clusters by third parties. ZB4OSGi fulfills to this feature by offering a mechanism that enables its extension, in order to include custom clusters in the abstraction of the ZigBee devices.

Integration mechanisms: ZB4OSGi maps the ZigBee devices with several OSGi services that export the ZigBee applications with high-level protocols (in the current implementation the UPnP and PERSONA exporters are available, see section V-c). The advantage is that external applications may access to the ZigBee devices by means of the mechanisms offered by high-level protocols, even if they differ from those used in the ZigBee network. For example the standard Temperature Sensor device only supports the pull access method (i.e. the applications should explicitly request for the value when they need it), and by means of the UPnP exporter such device can be extended with a push assess method (i.e. it may provide periodic readings to the interested recipients).

Management of different applications in the same ZigBee network: ZB4OSGi has been implemented on top of the OSGi platform to exploit its flexible service model. In this way ZB4OSGi can represent any APO or any ZigBee device (in accordance to the Abstraction Layer) in terms of an OSGi service. Such service can be easily controlled by exploiting the life cycle primitives provided by OSGi itself. In this way APOs from different ZigBee applications can

cohexist simultaneously within the OSGi execution environment.

B. Architectural Design

Figure 3 depicts the service-oriented model defined by the ZB4OSGi architecture. All of the 3 layers are here represented: Access, Abstraction, and Integration layer together with the underlying OSGi platform.

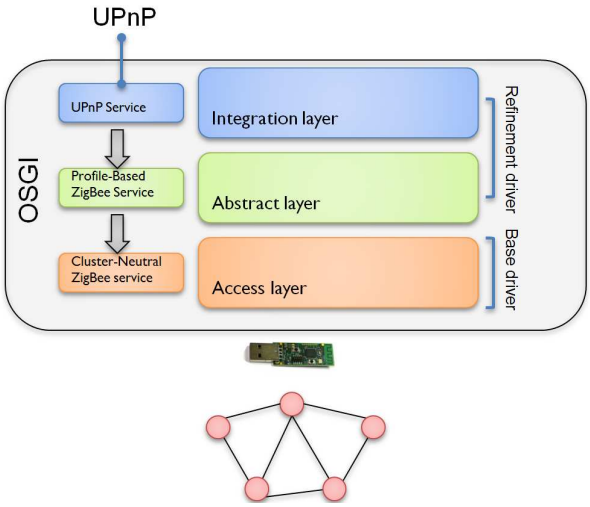

Fig.3. The ZigBee service model

The Access Layer directly communicates with the ZigBee network by means of a network adapter (called USB dongle), or by RS232 dongles or other kind of adapters.

According to the OSGi Device Access Specification [5], the component implementing the Access Layer is called Base Driver (in this work ZigBee Base Driver), while the components of the upper layers are called Refinement Drivers. The services registered by the Access Layer are gradually refined and further abstracted by means of the upper layers. In particular, the Access Layer registers a ZigBee service that is *cluster neutral*: it provides semantic-free methods that accept as formal parameter a ZigBee frame (represented as a sequence of byte) and inject the frame into the ZigBee network.

The Abstraction Layer introduces more semantic to the OSGi services, by refining them with new ones. These new OSGi services are dynamically registered according to the ZigBee profile implemented by the ZigBee devices. For this reason the Abstraction Layer registers ZigBee service that is *profile based*.

Note that, although the Abstraction Layer is designed as a generic layer, it should include a specific driver for each ZigBee profile in use in the ZigBee network. Since at the time of writing this paper, the most established cluster library is that of the Home Automation profile, thus the current implementation of ZB4OSGi only relies on the such profile. However, the development of further driver is also planned for other profiles as soon as their specifications will become public.

The Integration Layer, finally, maps the profile-based ZigBee services to an application-level protocol (the figure only reports the UPnP service). The way the Integration Layer reacts to the services registered by the Abstraction Layer, follows the standard OSGi event mechanism (see section II). This layer runs a set of exporters that detect the registration of profile-based ZigBee services. As soon as a new event occurs, such exporters will act as protocol translator by injecting the ZigBee devices into the appropriate network. For example, in a UPnP network the ZigBee devices can appear as virtual UPnP devices or they can appear as brand new SOAP end-point.

The next section describes in more details the Access Layer (namely ZigBee Base Driver), the Abstraction Layer and the Integration Layer (by only considering the UPnP exporter).

V. ZB4OSGi LAYERS

A. Access Layer

The Access Layer is implemented by means of the ZigBee Base Driver (ZBD). Its role is to introduce an initial abstraction of the ZigBee hardware by relying on a Hardware Abstraction Layer (HAL). The HAL aims at integrate heterogeneous ZigBee stack implementations, by freeing ZB4OSGi from any industry-driven solution. Together with such hardware abstraction, the Access Layer also provides a raw OSGi service for each ZigBee APO.

Figure 4 provides an overall view of the Access Layer, here the ZBD provides the ZigBee Device API (used by the Abstraction Layer) and it uses the Simple Driver API to abstract from a specific ZigBee hardware.

The ZigBee Device API models the notion of ZigBee Node in terms of network attributes like IEEE address, network address, node type, pan ID etc and the notion of End Point in terms of ZigBee attributes like profile ID, input cluster ID, output cluster ID, endpoint ID, device category etc. Moreover the ZigBee Device API also models the notion of ZigBee Cluster described in terms of cluster ID

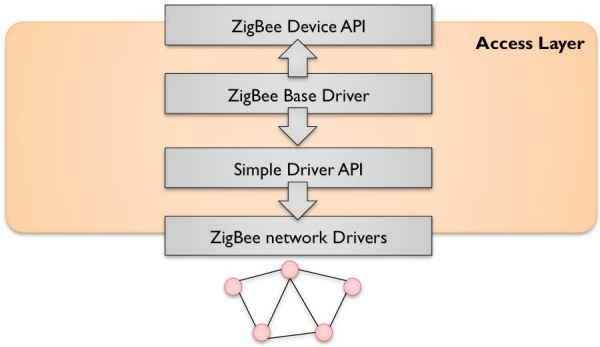

Fig. 4. The Access Layer

and the cluster message. The ZBD instantiates the ZigBee Device API as soon as it discovers one EP provided by the ZigBee network (the EP discovery mechanism is also described along this paragraph).

For each EP, it creates and registers an OSGi service called *ZigBeeDevice*. The *ZigBeeDevice* service acts as proxy for the EPs. In particular, when an application interacts with a *ZigBeeDevice* service, the ZBD forwards the messages to the corresponding EP on the ZigBee network. On the other hand, messages coming from the EPs are forwarded to the applications waiting for them (in our case ZBD forwards the message to the Abstraction Layer that, in turns, forwards the message to the high-level application by means of the Integration Layer).

The Simple Driver API (which is also a contribution of this work) defines a hardware interface industry-

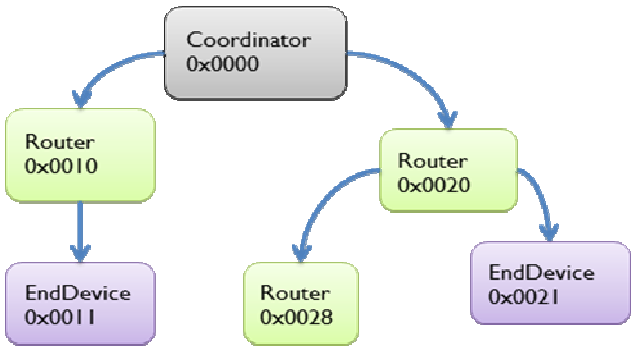

Fig. 5a. Example of addressing tree

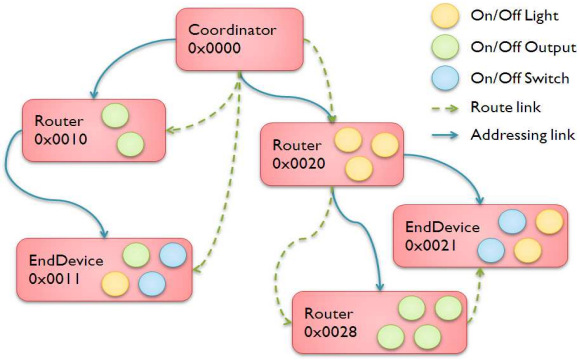

Fig. 5b. Example of ZigBee device discovery

independent that includes most of the common mechanisms needed to interact with a ZigBee network. The Simple Driver APIs are implemented by the ZigBee network Drivers that directly interact with the ZigBee network (section VI reports our case study, with some information about the CC2480 network driver). The Simple Driver API has been designed by taking into account a set of high-level primitives for the interaction with the network:

1) Create/join the network: this includes operations that configure the dongle in the ZigBee network by specifying the channel to use, the security key and the network identifier (pan ID). Moreover it is possible to interact with the dongle, to create a new network or to join an existing one.
2) Inspect the ZigBee node: by means of these operations, the ZigBee Base Driver obtains the IEEE address of the node, the list of EPs available on the node and the description of each EP.
3) Binding to an EP: this enables the ZigBee Base Driver to bind the dongle with one or more EPs and vice-versa or to bind two remote EPs with each other without the enrollment of the dongle.
4) Receive/send messages from/to an EP: these operations enable the ZigBee Base Driver to send messages to an EP or to receive messages from the EP (the messages are here represented always as array of bytes). The communication with an EP may follow a synchronous or asynchronous pattern. In the first case the operation (either send or receive) is implemented by means of a conventional synchronous Java method invocation, while in the latter case the communication is part of a more complex communication flow implemented by means of events and listeners pattern.
5) Inspect the status of the dongle: these operations provide methods to access the configurations of the dongle, like the assigned IEEE network address, the channel in use etc. Such configurations exploit the management and monitoring interface of the ZigBee stack.

Discovery mechanisms

In order to expose the EPs to the upper layers, the ZBD should be able to browse and to monitor the ZigBee network. The discovery algorithm used by the ZDB consists of two phases. The first phase collects the list of the ZigBee nodes available on the network; the second phase inspects the available EPs on the ZigBee nodes. Furthermore the ZBD should periodically run the first phase, in order to consolidate the current network topology. The next two subsections describe the first and second phase.

Discovery of ZigBee nodes

In ZigBee the mechanisms for browsing and monitoring the network are often not proactive (proactive notification mechanisms are only optional). For this reason the ZBD can discover the ZigBee nodes with two different strategies: *periodical browsing* and *continuous browsing*. The periodical browsing exploits only the ZigBee clusters that are mandatory for the ZigBee compliant devices. This ensures the compatibility of ZB4OSGi gateway with all ZigBee networks. The periodical browsing exploits the addressing-tree defined bythe ZigBee standard. It draws a logical tree (see Figure 5a) rooted on the coordinator of the network, where the intermediate nodes act as routers and the leaves as end-devices or routers. The addressing-tree is browsed by means of the *IEEE_addr_req* and *IEEE_addr_rsp* messages (mandatory in the IEEE standard). The former message requests for the IEEE address of a node together with the network address list of all the nodes "connected" to it. The second message carries the answer. The periodical browsing starts form the network coordinator (whose address is always 0x0000). The result is a very accurate topological perspective of the network, but it introduces a non-negligible network overhead. A reasonable configuration is to execute the periodic browsing unfrequently (for example every hour).

The continuous browsing uses optional ZigBee clusters offering an automatic notification mechanism for the connection of new nodes in the network, and for this reason it may result more efficient than the periodic browsing. Specifically, this method relies on the *Device_annce* message, which is only optional in the standard. This message is broadcasted from a ZigBee node as soon as it joins the network.

Discovery of EPs and ZigBee devices

When the ZBD detects a new ZigBee node, it further inspects it by fetching all the relevant information. ZBD exploits two pair of mandatory messages: *Active_EP_req*, *Active_EP_rsp* and *Simple_Desc_req*, *Simple_Desc_rsp*. The first pair of messages is used to retrieve the list of ZigBee EP available on the node. The second pair of messages retrieves the list of the clusters available on a specific EP. Once the inspection of a ZigBee node is completed, the ZBD registers an OSGi service for every EPs found. Figure 5b shows an example in which the ZBD browses a ZigBee network made of 6 nodes. Except the

coordinator, these nodes implement On/Off Light and On/Off Switch. Note that the solid lines in the figure define the network links among the nodes on the routing tree, while the dotted lines represent the sequence in which the ZBD queries the nodes.

B. *Abstraction Layer*

The role of the Abstraction Layer is to execute the drivers for each ZigBee profile in use in the ZigBee network. The Abstraction layer provides two features that distinguish ZB4OSGi from the existing solutions described in Table I:

1) provides an extensible mechanism for the integration of standard and custom ZigBee devices.
2) Reduces the complexity of accessing the ZigBee devices by offering easy-to-use APIs to the high-level applications.

Figure 6 depicts an overall view of the Abstraction Layer. The HA Profile Driver defines a set of hierarchical Java classes modeling the Home Automation devices here called Refinement Drivers. Examples of such drivers are: On/Off Switch, Remote Control, Door Lock, On/Off Light, Light Sensor. The HA drivers inspect all the *ZigBeeDevice*, registered by the Access Layer (more precisely it inspects the *ProfileID*, *DeviceID* and *ClusterID-List* of the *ZigBeeDevice*) in order to select the proper Device Factory. The role of the Factory is to create an independent mechanism for the selection and instantiation of the most suitable Refinement Driver for a raw *ZigBeeDevice* service. The Factory further verifies that the *ZigBeeDevice*

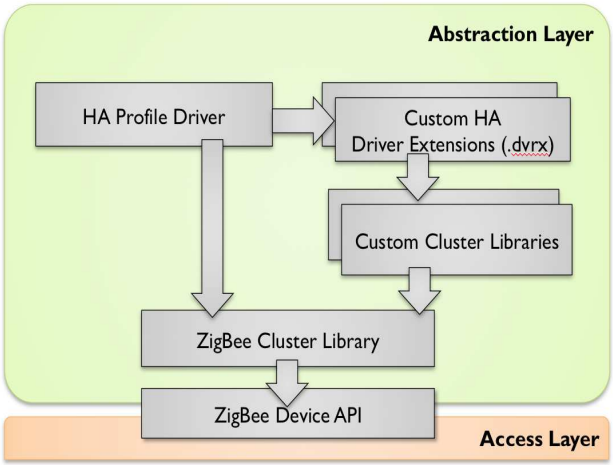

Fig. 6. The Abstraction Layer

implements all the mandatory clusters defined by the ZigBee profile. These mechanism enable ZB4OSGi to be a suitable platform not only for ZigBee devices adhering to a set of standard profiles, but also for those devices that implement a custom behavior. In fact, third-party developers may only provide their own implementation of the Device Factory together with the custom Refinement Driver, and then they may exploit the internal mechanisms of the Abstraction Layer for the dynamic instantiation of the proper driver.

The HA Profile Driver uses the ZCL bundle, that implements all the mandatory ZigBee clusters defined by the ZigBee profiles. Note that each cluster defines a set of attributes and a set of commands to update the value of the attributes. The clusters can inherit common commands that provide general-purpose operations; such commands have been accordingly factorized with the Attribute Interface in order to be defined only once and used by all the cluster definition.

The design of ZB4OSGi makes easy the interaction with the ZigBee devices by hiding the internal details of the ZigBee protocol. For this reason, the HA Profile Driver bundle also implements *glue code* that redefines the mandatory clusters with simpler versions. For example, let us consider a high-level application willing to be notified of the status change of the On/Off Light Device. By adopting the clusters defined in the ZCL (namely the *OnOff* cluster), the application should be aware and should manage many aspects of the ZigBee frames, like the ZigBee frame format or know the specific EP to which send the report message. All this complexity can be avoided by using a simplified version of the cluster where the caller has only to specify a listener object: *subscribe(OnOffLister listener).* As soon as the value changes, the listener, specified as parameter, is called back. The current implementation of ZB4OSGi offers simplified version of clusters for a large number of standard ZigBee clusters.

C. *Integration Layer*

The role of the Integration Layer is to export the ZigBee devices into different networks. ZB4OSGi provides a general-purpose solution, without introducing any specific constraint to the target network. This goes beyond the existing solutions reported in Table I, where the gateways embrace basically one exporting technology

The OSGi bundles implementing the business logic for the exporting procedure are called *exporters*. Currently, we have implemented two different exporters: UPnP exporter and PERSONA exporter [7] but other kinds of exporters can be easily integrated, e.g. Bluetooth network exporter, SOAP/REST exporter or Konnex exporter.

Since the Integration Layer is designed without considering a specific target technology, the rest of this section only focuses on the UPnP exporter as meaningful example of the integration between the ZigBee and UPnP network.

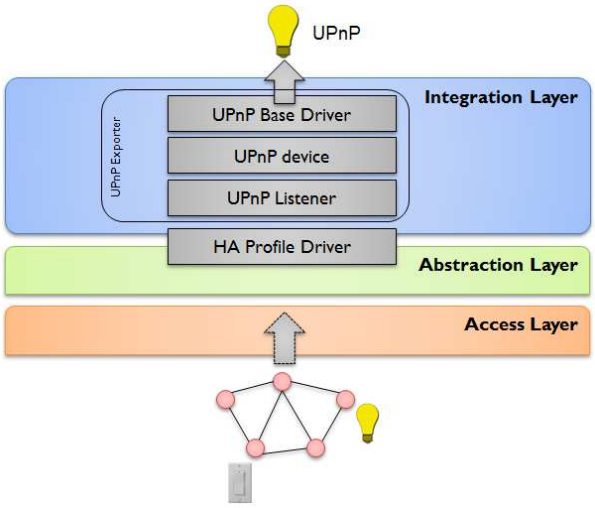

Fig. 7. The Integration Layer

Figure 7 depicts the UPnP exporter design. It is composed

of two core OSGi bundles: the UPnP Listener and the UPnP Base Driver. The UPnP Listener waits for newly registered Refinement Drivers and instantiates, for every driver, a *UPnP Device Service*. The *UPnP Device Service* wraps the Refinement Driver with an OSGi service implementing the standard UPnP interfaces (refer to *org.osgi.service.upnp* official UPnP APIs). More precisely, the role of the UPnP Listener is to map all the commands provided by the cluster's Refinement Driver as UPnP actions and, similarly, to map all the ZigBee attributes as UPnP state variables. In order to better clarify this mapping, Table II reports the association between UPnP state variable and ZigBee Attributes, and between UPnP actions and ZigBee commands for the standard On/Off Light Device.

TABLE II-A
MAPPING BETWEEN UPNP STATE VARIABLES AND ZIGBEE ATTRIBUTES

UPnP State Variable	ZigBee Attribute
Target	OnOff
Status	-

TABLE II-B
MAPPING BETWEEN UPNP ACTIONS AND ZIGBEE COMMANDS

UPnP Action	ZigBee Command
SetTarget(0)	Off
SetTarget(1)	On
GetTarget	Read attribute command
GetStatus	-

The *UPnP Base Driver* listens for the *UPnP Device Services* and injects them into the UPnP network. As soon the UPnP Base Driver exports a new UPnP device, the announce procedure of the UPnP protocol starts. All the existing UPnP control points can discover the brand new devices and start interacting with them in a standard way.

VI. CASE STUDY

This section describes the experiments that we performed to test ZB4OSGi gateway with a real ZigBee application. We have adopted Texas Instruments hardware in particular the EZ430-RF2480 demonstrator Kit. The kit relies on a USB dongle acting as network entry point, more specifically the CC2480 System on Chip (SoC). We configured a small, star-connected ZigBee network composed by two ZigBee nodes and the dongle acting as network coordinator. The dongle is connected to the ZB4OSGi stack running on top of a 3Ghz Windows XP PC.

It is important to remark that:

1) changes to the ZigBee hardware only requires to implement a new network driver implementing the Simple Driver API, without affecting the design of ZB4OSGi.
2) Adopting ZigBee profiles that differ from the Home Automation can be accomplished by implementing a new Factory and the Refined Drivers adhering to the profile.
3) The USB dongle can play the role of network coordinator or network end-point.

The ZigBee nodes have been configured in order to run ZigBee devices adhering to the Home Automation Profile. For this purpose we have compiled the Home Automation samples shipped with the ZStack© from Texas Instruments (we choose On/Off Light Device and On/Off Light Switch devices). By adopting third-party ZigBee devices, this experiment certifies the adherence of ZB4OSGi to the ZigBee profile definition.

The ZigBee devices are repeatedly queried by an external application (also running on the host PC) by means of the ZB4OSGi gateway. The requests issued to the ZigBee network vary in order to test different functionalities of the ZB4OSGi. The tests also give some results in terms of responsiveness of the whole system. From these tests it resulted that the system bottleneck is, by far, the EZRF430-CC2480 dongle. In particular, Figure 8 shows the average requests per second that the application of the host receives for different frequencies of the queries injection. These average refer to a set of tests in which, for each given frequency, 100 requests are issued by the application according to that frequency. The figure shows clearly that, for very low injection frequencies, the number of requests served per second scales linearly. However, the system reaches soon the saturation.

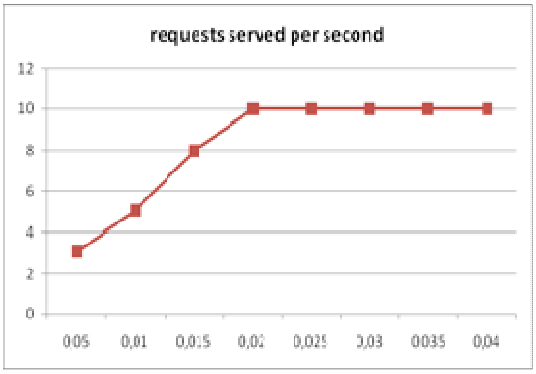

Fig. 8. Average number of requests served per second

This is due to the limits of the dongle that cannot sustain even moderate frequencies of messages. In fact, from a more detailed analysis, it resulted that the delay in serving a request is due only to less than 1% to the ZB4OSGi gateway, while the dongle accounts for almost 55% of the delay and the remaining part is due to delays introduced in the ZigBee network.

VII. CONCLUSION

The ZB4OSGi gateway enables the interconnection of ZigBee devices with external applications. Differently than the design of other ZigBee gateways, ZB4OSGi makes use of abstraction mechanisms that enable the applications to interact with the ZigBee network at high level, without any prior knowledge of the low level protocols and message formats of ZigBee. Furthermore ZB4OSGi has been designed considering an efficient extension mechanism for those ZigBee devices not adhering to any specific ZigBee profile.

At the current status, ZB4OSGi defines a driver for the Home Automation Profile of ZigBee, which is the first profile delivered and the most established one, however the implementation of drivers for new profiles is planned for the future. Further works are also scheduled for the extension of the supported ZigBee network drivers, more specifically support for the new CC2531 USB dongle and support for ember® ZigBee stack. It is intention of the ZB4OSGi team to enable the gateway to interact simultaneously with more than one single ZigBee dongle.

The overall architecture of ZB4OSGi demonstrates a mature degree of stability, currently it has been adopted within the framework of some European projects [19, 20] with remarkable results.

REFERENCES

[1] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y.F. Hu: Wireless Sensor Networks: a Survey on the State of the Art and the 802.15.4 and ZigBee Standards. In: Computer Communications, 30 (7): 1655-1695 (2007)

[2] S. Chessa: Sensor Network Standards book chapter in: J. Zheng and A. Jamalipour, Wireless Sensor Networks: A Networking Perspective, Wiley-IEEE Press, ISBN: 978-0-470-16763-2, September 2009, pp.407-431.

[3] ZigBee alliance, http://www.zigbee.org

[4] ZigBee Alliance, ZigBee Cluster Library Specification, http://www.zigbee.org, 2010

[5] The Open Source Gateway Initiative, http://www.osgi.org

[6] Universal Plug and Play, http:www.upnp.org

[7] The Persona middleware, http://www.persona.org

[8] ZigBee Alliance, Understanding ZigBee Gateway, http://www.zigbee.org, September 2010, pp.1-16.

[9] OSGi Service Platform Release 4, Version 4.1, May 2007 (http://www.osgi.org).

[10] C. Lee, D. Nordstedt, and S. Helal, “Enabling smart spaces with OSGi”, IEEE Pervasive Computing 2(3):89-94, 2003.

[11] ZigBee Alliance, ZigBee specification, http://www.zigbee.org, 17 January 2008.

[12] Woo Suk Lee; Seung Ho Hong, “Implementation of a KNX-ZigBee gateway for home automation”, IEEE 13th International Symposium on Consumer Electronics, pp. 545 – 549 (2009)

[13] Kawamoto, R.; Emori, T.; Sakata, S.; Furuhata, K.; Yuasa, K.; Hara, S., “DLNA-ZigBee Gateway Architecture and Energy Efficient Sensor Control for Home Networks”, 16th IST Mobile and Wireless Communications Summit, pp. 1-5 (2007).

[14] Seong Hoon Kim; Jeong Seok Kang; Hong Seong Park; Daeyoung Kim; Young-joo Kim, “UPnP-ZigBee internetworking architecture mirroring a multi-hop ZigBee network topology”, IEEE Transactions on Consumer Electronics, 55 (3): 1286 – 1294 (2009)

[15] Guozhen Hu, “Design and implementation of industrial wireless gateway based on ZigBee communication”, 9th Int. Conf. on Electronic Measurement & Instruments (ICEMI) 2009, pp. 1-684 – 1.688

[16] Peng Qiu; Ung Heo; Jaeho Choi, “The web-sensor gateway architecture for Zigbee”, IEEE 13th Int. Symp. on Consumer Electronics (ISCE), pp. 661 – 664 (2009)

[17] De Silva, G.S.H.; De Silva, L.W.R.; Ishara, P.W.K.; Kumara, M.P.H.; Ginige, T., “SmartBee; Multichannel Access ZigBee Gateway with Plug and Play Device Interface for Smart Home/Office Automation”, 4th Int. Conf. on Information and Automation for Sustainability (ICIAFS) 2008, pp. 251 – 256

[18] Young-Guk Ha, “Dynamic integration of zigbee home networks into home gateways using OSGI service registry”, IEEE Transactions on Consumer Electronics, 55 (2) : 470 – 476 (2009)

[19] EU FP7 project PERSONA : Perceptive spaces promoting independent aging, http://www.aal-persona.org/, 2006-2009

[20] EU FP7 project universAAL: UNVERSal open platform and reference architecture Specification for Ambient Assisted Living, http://www.universaal.org, 2010-2014

[21] Ambient Assisted Living Open Association (AALOA), http://www.aaloa.org, 2011

[22] The ZigBee4OSGi project, http://zb4osgi.aaloa.org/, 2011

[23] Marshini Chetty, Ja-Young Sung, and Rebecca E. Grinter. 2007. How smart homes learn: the evolution of the networked home and household. In Proceedings of the 9th international conference on Ubiquitous computing (UbiComp '07), John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang (Eds.). Springer-Verlag, Berlin, Heidelberg, 127-144.

[24] Fung, B.; Trojer, T.; Hung, P.; Xiong, L.; Al-Hussaeni, K.; Dssouli, R.; , "Service-Oriented Architecture for High-Dimensional Private Data Mashup," Services Computing, IEEE Transactions on , vol.PP, no.99, pp.1, 0 doi: 10.1109/TSC.2011.13